# CRC

Basic idea: [illegible] by [illegible] the CRC [illegible] value.

[illegible] $\rightarrow x^{12} + x^{8} + x^{4} + x^{2}$

[illegible] $C$ [illegible]

$M$ [illegible]

$\rightarrow$ [illegible] $G$ [illegible]

[illegible] $\frac{M}{G} = Q$ rem $R$ [illegible]

$M$ [illegible] $G$ [illegible]

equivalently: $\frac{M}{G} = Q + \frac{R}{G}$ or $M = GQ + R$

From this, we can see that $M-R$ will be [illegible]:

$$M-R = GQ + R - R$$
$$M-R = GQ$$
$$\frac{M-R}{G} = Q \text{ rem } 0$$

If we transmit $M-R$, the receiver can divide by $G$: if it divides evenly, we assume there were no errors. If there is a non-zero remainder, there must have been an error.

However, $M$ is not recoverable from $M-R$ unless we know $R$, & we [illegible] know $R$ unless we know $M$. Therefore [illegible] [illegible], [illegible] we haven't actually transmitted the message

[illegible] would like to [illegible] both $M$ and $R$, the [illegible] the [illegible] $M$-$R$ [illegible]

[illegible] $M$ & $R$ together [illegible] receiver [illegible]

[illegible]

Summary:

Assume [illegible] bits

[illegible] by [illegible]

[illegible] bits w/ value [illegible] $M$ [illegible]

[illegible] $E$ [illegible] $E =$ [illegible]

[illegible] $E$ [illegible] $G$ [illegible]

[illegible] Quotient $Q$ is [illegible]

[illegible] $E$ [illegible]

$C = E - R$ $(M \cdot 2^{w}$ [illegible]

Send $C$

As we already saw, $C$ will necessarily [illegible] by [illegible] [illegible] $C = E + R$. [illegible] will not have [illegible] significant bits [illegible] $E = M2^{w}$ [illegible] $M$ w/ $R$ [illegible]

The receiver therefore checks for errors [illegible] if the received version of C, $\tilde{C}$, is evenly divisible by G. If it is, they assume no errors ($\tilde{C} = C$), & can recover M by subtracting R & dividing by $2^w$.

$$M = \frac{(C - R)}{2^w}$$

Or equivalently, they just cut off the last w bits.

Note that it is not [illegible] by G, [illegible] be [illegible] of $\frac{M 2^w}{G}$, which is [illegible] the same.

[illegible]

In this arithmetic, [illegible] must not be concerned for [illegible], because [illegible] and [illegible] the same. For instance:

$$1011 + 101 = 1110$$
$$1011 - 101 = 1110$$

[illegible] 1001 − 110 = 1111 [illegible] 1111. On the other hand, [illegible] cannot [illegible].

[illegible] are [illegible]

$$1011 \overline{)1110} \quad \ldots = 1 \text{ rem } 101$$
$$-101$$
$$0101$$

$$1110 \overline{)1011} \rightarrow \frac{1011}{1110} = 1 \text{ rem } 101$$
$$-1110$$
$$0101$$

[illegible] number [illegible] fewer significant [illegible] always is than [illegible]

[illegible] > 101

[illegible]

[illegible]

[illegible] 1 0 0 0 1 0

10 [illegible]

0011

[illegible]

000101

0 0 1 0 1 0

0 0 0 0 [illegible]

0 1 0 1 0 0

1 0 0 1 1

0 0 1 1 1 0

0 0 0 0 0

0 1 1 1 0

recall [illegible] don't [illegible] about the quotient, only the remainder. Take a look at the steps we followed during the division, specifically [illegible] to the [illegible].
At each step, if the remainder's highest order bit [illegible] spot of the divisor (they have the same number of significant bits), then the divisor [illegible] goes into [illegible] once, [illegible] and [illegible].
[illegible] hand, if the remainder's highest bit is [illegible] than the divisor's, [illegible] goes [illegible] times, [illegible]

Then, [illegible], we shift the next bit [illegible] remainder.

[illegible] the [illegible] array of bits $M$, & the remainder [illegible]

```
R=0;
for (bit in M) {
  if (R >> (W-1)) [illegible]
     R = R ^ G;
  }
  R = (R << 1) | bit;
```

[illegible] more straightforward [illegible] we need to shift $M$ up by [illegible] bits before [illegible]

## Table based implementation

[illegible] process 2 bits of the message of our [illegible] i.e.

$\boxed{a_4 \ldots a_1\ a_0}$ Having a [illegible]

When we're [illegible] message bit, $m_i$, [illegible] determines whether or not we XOR $G$ into the [illegible]?

$$\begin{array}{cc} & \boxed{a_3 \mid a_2 \mid a_1 \mid a_0 \mid m_i} \\ + & (a_4 * \boxed{g_4 \mid g_3 \mid g_2 \mid g_1 \mid g_0}) \end{array}$$

$$\boxed{a_3+(a_4g_4) \mid a_2+(a_4g_3) \mid a_1+(a_4g_2) \mid a_0+(a_4g_1) \mid m_i+(a_4g_0)}$$

We'll call these new values $\boxed{b_4 \mid b_3 \mid b_2 \mid b_1 \mid b_0}$

When [illegible] next bit, it will be $b_4 = a_3+(a_4g_4)$ which determines whether or not we XOR.

$$\begin{array}{cc} & \boxed{b_3 \mid b_2 \mid b_1 \mid b_0 \mid m_{i+1}} \\ + & (b_4 * \boxed{g_4 \mid g_3 \mid g_2 \mid g_1 \mid g_0}) \end{array}$$

$$\boxed{c_4 \mid c_3 \mid c_2 \mid c_1 \mid c_0}$$

Notice that $c_0 = m_{i+1} + (b_4 g_0)$
$= m_{i+1} + ((a_3+(a_4g_4))g_0)$

& $c_1 = b_0 + (b_4 g_1)$
$= m_i + (a_4 g_0) + ((a_3+(a_4g_4))g_1)$

Similarly $c_2 = b_1 + (b_4 g_2)$
$= a_0 + (a_4 g_1) + ((a_3+(a_4g_4))g_2)$

$$C_3 = b_2 + (b_4 g_3)$$
$$= a_1 + (a_4 g_2) + ((a_3 + (a_4 g_4)) g_3)$$

and

$$C_4 = b_3 + (b_4 g_4)$$
$$= a_2 + (a_4 g_3) + ((a_3 + (a_4 g_4)) g_4)$$

So you can see now that after shifting in a message bits, each of our final [illegible] is of the form:

$$C_j = x_j + (a_4 g_{j-1}) + ((a_3 + (a_4 g_4)) g_j)$$

where $x_0$ is $m_{i+1}$, $x_1$ is $m_i$, & $x_j = C_{j-2}$ for $2 \le j \le 4$ & note that $g_{-1}$ (used for $C_0$) is simply 0.

This can be further "simplified" as

$$C_j = x_j + T_j$$

where $T_j$ is [illegible] a ($a_4$ & $a_3$) [illegible]

So [illegible], we have:

$$\begin{array}{rl} & \boxed{a_2 \mid a_1 \mid a_0 \mid m_i \mid m_{i+1}} \\ + & \boxed{\qquad\quad T \qquad\quad} \\ \hline & \boxed{c_4 \mid c_3 \mid c_2 \mid c_1 \mid c_0} \end{array}$$

Recall that $T$ is a function [illegible] $a_3$ & $a_4$ [illegible] $g_j$. That means we can precompute $T$ for all possible values [illegible] $a_4$ & $a_3$ [illegible] we need [illegible] shift [illegible] message [illegible] from the [illegible], [illegible] $T(a_4\ a_3)$ [illegible] ...

[illegible] more [illegible] [illegible]
a four [illegible] of [illegible]
[illegible]

[illegible] 10K [illegible]
[illegible]

[illegible] made [illegible]
[illegible] same size as
[illegible]

[illegible] that C can [illegible]
[illegible]

Computing the table

We can see by [illegible] that [illegible] computing $T_g(x)$ is [illegible] the top $S$ bits of $x$, [illegible] $x$, and [illegible] with 0, [illegible]

$$T_g(x)$$

[illegible] $S=3$ & $x = a_2 2^2 + a_1 2 + a_0$:

Start with $|a_2|a_1|a_0|0|0|$

then do 3 iterations: $|a_1|a_0|0|0|0|$

$\oplus\ (a_2 * g)$

$|b_2|b_1|b_0|g_1|g_0|$

$\Downarrow$

$|b_1|b_0|g_1|g_0|0|$

$b_2 * g$

$|c_2|c_1|c_0|g_0g_1|g_0|$

$\Downarrow$

$|c_1|c_0|g_0g_1|g_0|0|$

$c_2 * g$

$T_g(a_2 2^2 + a_1 2 + a_0) \rightarrow |d_2|d_1|d_0|g_0g_1|g_0|$

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

so we can [illegible] the algorithm [illegible] of [illegible] [illegible]
[illegible] values
then shifting [illegible] [illegible], we can just
XOR the next 5 [illegible] the message with the [illegible]
[illegible]
$g$.
So [illegible]

No [illegible]

that [illegible] message data [illegible]
instead [illegible], so we [illegible]
skip [illegible] point at this [illegible]
skip [illegible]

[illegible]

[illegible] before [illegible]

[illegible]

[illegible]

[illegible] used [illegible] [illegible] [illegible] called the [illegible] value.

[illegible] bit is 1.

E.g. [illegible] value [illegible] (modified)

[illegible] 1000 [illegible] 1001 [illegible] 0010100 [illegible]

[illegible]